IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application for

METHODS AND APPARATUS FOR REMOTE PROCESS CONTROL

Appendix I

(source code listings)

```
//
// psap.h
// The Foxboro Company Confidential
// Copyright (c) The Foxboro Company.  All Rights Reserved
// G. Couper  8/14/96
//
/*
This file is intended to compensate for not using ${IA}/header/om_user.h
and all of the things that conflict with JAVA.
*/

#include <sys/types.h>

/*
typedef unsigned char   u_char;
*/

#define MAX_NSAP_LEN        20 /* maximum # bytes in net. addr.*/
#define MAX_SSAP_LEN         2 /* Max. # of bytes in  SSAP id  */

struct PSAP_ADDR {
    u_char ssap_id[MAX_SSAP_LEN];
    u_short tsap_id;
    u_short nsap_len;
    u_char nsap_address[MAX_NSAP_LEN];
    };

typedef struct PSAP_ADDR PSAP_ADDR;

typedef    PSAP_ADDR    *PSAP_ADDR_PTR ;

#define   PSAP_ADDR_FDRSIZE (MAX_SSAP_LEN+2+2+MAX_NSAP_LEN)

#define   PSAP_SIZE    sizeof( PSAP_ADDR )
```

```java
//
// ThreadedTrendServer.java
// The Foxboro Company Confidential
// Copyright (c) The Foxboro Company.  All Rights Reserved
// G. Couper, B. Canna  8/14/96
//

import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

public class ThreadedTrendServer extends Thread
{
    static // try to load the library of native methods
    {
        try
        {
            System.out.println("loading library");
            System.loadLibrary("trendm");
        }
        catch (UnsatisfiedLinkError e)
        {
            System.out.println("Can't find library trendm");
            System.exit(-1);
        }
    } //end of static library load


    native    int omlistcreate    ( );                        //creates a omlist
    native    int omlistadd       ( String varname );         //adds a point to the list
    native    int omlistopen      ( );                        //opens the list
    native    int omlistgetupdate ( byte b[], int len );      //calls dqchange without suspending
    native    int omlistclose     ( );                        //closes the list


    int              threadnum;
    byte             update[];
    boolean          updating = false;
    Socket           sock;
    DataInputStream  sockIn;
    DataOutputStream sockOut;
    String           line;


    // The following members declare space used by the native methods//
    int  jlistpoints;         // C library keeps the number of points on the OM list
    int  jin_open_id;         // C library stores the Open Id into here
    byte jin_om_desc[] ;      // C library stores om_header node in here
    byte jin_var_list[] ;     // C library stores the open var list in here
    byte jin_net_adr_tbl[];   // C library stores the net address table in here
    byte jin_data[];          // C library dqueues changes into here
    // The preceding members declare space used by the native methods//


    ThreadedTrendServer( Socket s, int c )
    {
        sock      = s;
        threadnum = c;
        update    = new byte[225];               // setup the update buffer
    }


    public void run()                            // this method is started by .start on the thread class.
    {
        try
        {
            // get input and output streams associated w/ socket
            sockOut = new DataOutputStream(sock.getOutputStream());
            sockIn  = new  DataInputStream(sock.getInputStream());

            jin_om_desc     = new byte[320];     // allocate the real space in Java for C library
state
            jin_var_list    = new byte[320];
            jin_net_adr_tbl = new byte[320];
            jin_data        = new byte[320];
```

- Appendix Page 2 -

```java
// poll for messages from client and OM changes until client
// disconnects (via OMBREAK command)
while (true)
{
    try
        this.sleep(1000);  // sleep for a sec so other threads can run.
    catch( InterruptedException e ) {}

    if ( 0 < sockIn.available() )
    {
        line = sockIn.readLine();      // get a line from the socket

        //print the line for checking purposes
        System.out.println(threadnum + "> trendserver: received: " + line );

        // were we asked to OPEN the list?
        if( line.startsWith("OMOPEN") )
        {
            String name;
            StringTokenizer st = new StringTokenizer(line, " ;=");

            //print the line for checking purposes
            System.out.println(threadnum + "> trendserver OMOPEN: request recognized" );

            System.out.println(threadnum + "> trendserver OMOPEN: creating omlist...");
            omlistcreate();


            // First - get rid of the OMOPEN token
            name = st.nextToken();

            // Now - get each name on the OMOPEN line
            while ( st.hasMoreTokens() )
            {
                // Should be NAME token
                name = st.nextToken();
                System.out.println(threadnum + "> trendserver OMOPEN: adding '" + name + "'
to the list..." );

                omlistadd(name);
            }


            System.out.println(threadnum + "> trendserver OMOPEN: opening the list...");
            omlistopen( );

            updating = true;

        } // end of OMOPEN

        // if we are asked to CLOSE the list
        else if (line.equals("OMCLOSE"))
        {
            System.out.println(threadnum + "> trendserver: close command recognized.
Closing the OM lists...");
            omlistclose ();
            sockOut.writeBytes( "OMCLOSEOK\n" );  // sends close string to socket
            updating = false;

        } // end of OMCLOSE

        // if we are asked to BREAK the connection
        else if (line.equals("OMBREAK"))
        {
            System.out.println(threadnum + "> trendserver: break command recognized.
Closing the connection...");
            omlistclose ();
            updating = false;
            sock.close();
            break;
        } // end of OMBREAK
    }

    if ( updating )
    {
```

- Appendix Page 3 -

```
        int numchars = omlistgetupdate( update , 225 );
        if (numchars > 0) System.out.print( threadnum + "> " );
        for(int i=0;i<numchars;i++)
        {
            System.out.print( (char) update[i] );  // sends it to the server console
            sockOut.write(      (int) update[i] );  // sends it to socket
        }
    }
    } // while loop for update
    } //end of run try

    catch(IOException e)
    {
        System.out.println("\r" + threadnum + "> trendserver: exception (client disconnected).
Continuing...");
        System.out.println(         threadnum + "> trendserver: closing OM list...\r");
        omlistclose ();
        try
            sock.close();
        catch( IOException ex ) {}
    } //end of run catch
    } //end of run
} // end of ThreadedTrendServer
```

```
//
// ThreadedClient.java
// The Foxboro Company Confidential
// Copyright (c) The Foxboro Company.  All Rights Reserved
// A. Nauman, B. Canna  8/14/96
//
import java.util.StringTokenizer;
import java.awt.*;
import java.awt.image.*;
import java.net.*;
import java.io.*;
import java.lang.*;
import java.applet.*;


/////////////////////////////////////////////////////////////////
//   CLASS:    TrendClient
//
//   FUNCTION:
//   -- initialize the applet
//   -- add new panel at the bottom of the applet frame for
//      entering points of interest and displaying their text
//      value
//   -- create three "OpenPoints" for possible use
//   -- create the "TrendCanvas" for displaying the trend lines
//   -- create the "TCPClient" to support communications with
//      the server program offering OM data
//
/////////////////////////////////////////////////////////////////
public class TrendClient extends Applet
{
    private TCPClient    tClient;
    private TrendCanvas  painter;
    private Button       startTrend;
    private Button       stopTrend;
    private OpenPoint    point1;
    private OpenPoint    point2;
    private OpenPoint    point3;

    public void init()
    {
      String portId = "";
      portId =  getParameter("portId");
      String hostId = "";
      hostId = getParameter("hostId");
      System.out.println( "host id is " + hostId + " port Id is " + portId );
      setLayout(new BorderLayout());
      Panel p = new Panel();
      p.setLayout(new GridLayout(4,2));
      p.add(startTrend = new Button("Start Trend"));
      p.add(stopTrend = new Button("Stop Trend"));
      add("South", p);
      point1 = new OpenPoint("Y14CP3_01:PID_10.OUT", Color.yellow, p );
      point2 = new OpenPoint("Y14CP3_07:PID_18.OUT", Color.white,  p );
      point3 = new OpenPoint("RAMP_Y14CP3:LEAD_Y14CP3.OUT", Color.green,  p );
      setBackground(Color.black);
      painter = new TrendCanvas(point1, point2, point3);
      add("Center", painter);

      try
      {
          tClient = new TCPClient(hostId, portId, this);
          if( tClient == null )
            System.out.println("Unable to create a connection to server");
          else
            tClient.setPriority(Thread.NORM_PRIORITY + 2);
      }
      catch (IOException e) {}
    }


/////////////////////////////////////////////////////////////////
// METHOD:   destroy
//
// FUNCTION:
//   -- respond to Netscape exit (i.e., client applet is going
//      DOWN!)
```

```
//
////////////////////////////////////////////////////////////////
public void destroy()
{
   try
     tClient.sSend("OMBREAK");
   catch( IOException e) {};
}


////////////////////////////////////////////////////////////////
// METHOD:   action
//
// FUNCTION:
//   -- respond to "STOP TREND" and "START TREND" requests from
//      the panel
//
////////////////////////////////////////////////////////////////
public boolean action(Event evt, Object arg)
{
  if (arg.equals("Start Trend"))
  {
    String message = "";

    point1.activate();
    point2.activate();
    point3.activate();

    if( point1.active )
        message = message + point1.name + " ";
    if( point2.active )
        message = message + point2.name + " ";
    if( point3.active )
        message = message + point3.name + " ";

    if( message != "" )
    {
        try
          tClient.sSend("OMOPEN " + message);
        catch( IOException e) {};
    }
  }
  else if (arg.equals("Stop Trend"))
  {
     try
        tClient.sSend("OMCLOSE");
     catch( IOException e) {};
  }
  else
    return false;
  return true;
}


////////////////////////////////////////////////////////////////
// METHOD:   newNVpair
//
// FUNCTION:
//   -- update the appropriate open point with a new value
//      received from the server offering OM data. Note that
//      all points invoked but only those that match the correct
//      name will be updated.
//
////////////////////////////////////////////////////////////////
public void newNVpair(String name, String value)
{
     point1.updateNV(name, value);
     point2.updateNV(name, value);
     point3.updateNV(name, value);
}


////////////////////////////////////////////////////////////////
// METHOD:   displayTrend
//
// FUNCTION:
//   -- update the trend lines on the TrendCanvas
//   -- will call the TrendCanvas "brush" method to do so!
```

```
//
///////////////////////////////////////////////////////////////////
public void displayTrend()
{
    painter.brush();
}


///////////////////////////////////////////////////////////////////
// METHOD:    clearTrendDisplay
//
// FUNCTION:
//   -- clear the trend lines on the TrendCanvas
//   -- zero out text values for points
//   -- typically called after an OMCLOSEOK has been received
//      from the server offering OM data
//
///////////////////////////////////////////////////////////////////
public void clearTrendDisplay()
{
    point1.deactivate();
    point2.deactivate();
    point3.deactivate();

    painter.clear();
}

}  // END OF Class TrendClient




///////////////////////////////////////////////////////////////////
//   CLASS:     TrendCanvas
//
//   FUNCTION:
//   -- supports the trend display
//   -- contains a grid AND
//   -- the ability to draw trend lines
//
///////////////////////////////////////////////////////////////////
class TrendCanvas extends Canvas
{
    private OpenPoint point1;
    private OpenPoint point2;
    private OpenPoint point3;
    private Image      bufferedImage = null;
    private boolean    clearDisplay = false;
    private int        max_y   = 300;
    private int        y_scale = 2;

    TrendCanvas( OpenPoint p1, OpenPoint p2, OpenPoint p3 )
    {
        point1      = p1;
        point2      = p2;
        point3      = p3;
    }


///////////////////////////////////////////////////////////////////
// METHOD:   paint
//
// FUNCTION:
//   -- draws the grid lines on the Trend display in a buffered
//      image for performance reasons
//   -- likewise for the trend lines
//
///////////////////////////////////////////////////////////////////
public void paint(Graphics g)
{
    bufferedImage = createImage(800,500);
    Graphics bg = bufferedImage.getGraphics();
    bg.setColor(Color.black);
    bg.fillRect(0, 0, 800, 500);

    // draw the X and Y axes as well as labels
    bg.setColor(Color.white);
```

```java
    bg.drawLine(100,50,100,400);                    // y-axis
    bg.drawLine(100,400,(max_y*y_scale)+100,400);   // x-axis
    int diff=140;
    for( int y=60; y<=400; y+=20, diff -= 40 )
    {
       String str2 = "" + (y+diff);
       bg.drawLine(90, y, 100, y );
       bg.drawString(str2,60,(y + 5));              // y-axis labels
    }

    bg.setFont(new Font( "Times Roman", Font.BOLD, 14));
    bg.setColor(Color.white);
    bg.drawString("Time (seconds)",350,450);        // x-axis title

    // now draw the trend lines
    if( clearDisplay == false )
    {
       drawTrendLine( point1, bg );
       drawTrendLine( point2, bg );         .
       drawTrendLine( point3, bg );
    }
    clearDisplay = false;
    bg.dispose();

    // now draw the image on the canvas
    g.drawImage( bufferedImage, 0, 0, Color.black, null );

}

//////////////////////////////////////////////////////////////////
// METHOD:    update
//
// FUNCTION:
//  -- overloads default update() method so that no automatic
//     screen erase occurs
//
//////////////////////////////////////////////////////////////////
public void update( Graphics g)
{
    paint(g);
}

//////////////////////////////////////////////////////////////////
// METHOD:    drawTrendLine
//
// FUNCTION:
//  -- Draw a trend line.
//  -- requires that data be passed to us via a circular buffer
//  -- ensure that the point is "active"
//  -- use the right color
//  -- draw a line from the last (x,y) coordinate to the current
//     (x,y) coordinate. If beginning to draw the line, then
//     don't draw from (0,0).
//
//////////////////////////////////////////////////////////////////
public void drawTrendLine( OpenPoint p, Graphics bg )
{
    int numEntries;
    int bufSize;
    int startIdx;
    int i;

    if( p.active )
    {
       p.updateHistory();

       bufSize    = p.bufSize;
       startIdx   = (p.numEntries >= bufSize ) ? p.startIdx : 0;
       numEntries = p.numEntries-2;

       bg.setColor(p.color);
       for( i = 0; i < numEntries; i++, startIdx++ )

          bg.drawLine(i*y_scale+101, 260-p.history[(startIdx%bufSize)],
                      i*y_scale+103, 260-p.history[((startIdx+1)%bufSize)]);
    }
```

- Appendix Page 8 -

```
    }
}

////////////////////////////////////////////////////////////////
// METHOD:    brush
//
// FUNCTION:
//    -- method to allow other objects to force redraw of trend lines
//
////////////////////////////////////////////////////////////////
public void brush()
{
    repaint();
}

////////////////////////////////////////////////////////////////
// METHOD:    clear
//
// FUNCTION:
//    -- method to allow other objects to force clearing of trend
//       lines
//
////////////////////////////////////////////////////////////////
public void clear()
{
    clearDisplay = true;
    repaint();
}

} // END OF Class TrendCanvas



////////////////////////////////////////////////////////////////
//   CLASS:    OpenPoint
//
//   FUNCTION:
//    -- contains information about each open point
//    -- allows other objects to activate the point (tell server
//       offering OM data that the point should be scanned)
//    -- allows other objects tp deactivate the point
//    -- allows other objects to set the lastValue (value of the
//       open point during the last time interval) to the current
//       value
////////////////////////////////////////////////////////////////
class OpenPoint
{
    public  String     name;       // Name of the open point
                                    // (HACK -- should be private)
    public  int[]      history;     // Circular buffer for open point values
                                    // (HACK -- should be private)
    public  int        bufSize;     // Size of circular buffer
                                    // (HACK -- should be private)
    public  int        numEntries; // Number of valid entries in the buffer;
                                    // (HACK -- should be private)
    public  int        startIdx;    // current index of buffer containing
                                    // newest data
                                    // (HACK -- should be private)
    public  Color      color;       // Color to display the open point
                                    // (HACK -- should be private)
    public  boolean    active;      // Is this point being trended?
                                    // (HACK -- should be private)
    private String     stringVal;   // Value of open point stored as a String
    private Panel      panel;       // pointer to the panel
    private TextField  field;       // field in the panel to hold the point name
    private Label      label;       // field in the panel to hold the point value

    public OpenPoint( String inputName, Color inputColor, Panel p )
    {
        name       = inputName;
        bufSize    = 300;
        history    = new int[bufSize];
        history[0] = 0;
        startIdx   = 0;
        numEntries = 0;
```

- Appendix Page 9 -

```java
color        = inputColor.brighter();
active       = false;
panel        = p;
stringVal    = "0";
p.setBackground(Color.darkGray);
p.setForeground(color);
p.add(field = new TextField(name, 4));
p.setBackground(Color.cyan);
p.add(label = new Label(stringVal));
}

/////////////////////////////////////////////////////////////
// METHOD:   activate
//
// FUNCTION:
//   -- sets active flag if there is a name in the NAME field
//
/////////////////////////////////////////////////////////////
public void activate()
{
   name = field.getText();
   if( name != "" )
     active = true;
}


/////////////////////////////////////////////////////////////
// METHOD:   deactivate
//
// FUNCTION:
//   -- unsets active flag; typically called in response to a
//      STOP TREND command
//
/////////////////////////////////////////////////////////////
public void deactivate()
{
   active      = false;
   stringVal   = "0";
   startIdx    = 0;
   numEntries  = 0;
   label.setText( stringVal );
}


/////////////////////////////////////////////////////////////
// METHOD:   updateNV
//
// FUNCTION:
//   -- updates the current and last values when an incoming
//      OMUPDATE message was received from the server offering
//      OM data
//
/////////////////////////////////////////////////////////////
public void updateNV( String n, String v )
{
   if( active && name.equals(n) )
   {
      stringVal = v;
      label.setText( stringVal );
   }
}


/////////////////////////////////////////////////////////////
// METHOD:   updateHistory
//
// FUNCTION:
//   -- updates the circular buffer with the latest value
//
/////////////////////////////////////////////////////////////
public void updateHistory()
{
   history[startIdx++] = Float.valueOf(stringVal).intValue();;
   startIdx %= bufSize;
   if( ++numEntries >= bufSize )
      numEntries = bufSize;
}
} // END OF Class OpenPoint
```

```
///////////////////////////////////////////////////////////////
//   INTERFACE:    Timed
//
//   FUNCTION:
//   --  define an "interrupt" method to be invoked on an arbitrary
//       event (in this case it will be a clock tick as defined
//       in the "Timer" class).
//
///////////////////////////////////////////////////////////////
interface Timed
{
   public void tick(Timer t);
}



///////////////////////////////////////////////////////////////
//   CLASS:    Timer
//
//   FUNCTION:
//   --  create a thread that wakes up every second (or so)
//       and invokes the tick "interrupt" call
//
///////////////////////////////////////////////////////////////
class Timer extends Thread
{
   private Timed    target;
   private int      interval;

   public Timer(Timed t, int i)
   {  target  = t; interval = i;
      setDaemon(true);
   }

   public void run()
   {  while (true)
      {  try { sleep(interval); }
         catch(InterruptedException e) {}
         target.tick(this);
      }
   }
}  // END OF Class Timer



///////////////////////////////////////////////////////////////
//   CLASS:    TCPClient
//
//   FUNCTION:
//   --  enable sending of messages to the server
//   --  start a timer thread that listens for OMUPDATE
//       messages
//   --  update the trend display with NEW values upon receipt
//       of OMUPDATE messages OR
//   --  update the trend display with the EXISTING values
//
///////////////////////////////////////////////////////////////
class TCPClient extends Thread implements Timed
{
   public  int              xcount = 0;
   private int              firstUpdate = 0;
   private DataInputStream  sIn;
   private DataOutputStream sOut;
   private Timer            t;
   private Socket           s;
   private TrendClient      frame;

   public TCPClient(String hostId, String portId, TrendClient f)throws IOException
   {
      frame = f;
      s = new Socket(hostId, Integer.parseInt(portId) );
      sIn = new DataInputStream(s.getInputStream());
      sOut = new DataOutputStream(s.getOutputStream());
      t = new Timer(this,1000);
      t.start();
```

- Appendix Page 11 -

```
}
/////////////////////////////////////////////////////////////////
// METHOD:   tick
//
// FUNCTION:
//   -- check the port for incoming data
//   -- if data available, then read a line of data and check
//      for OMUPDATE or OMCLOSEOK messages.
//
/////////////////////////////////////////////////////////////////
public void tick(Timer t)
{
   String line = "";
   try
   {
      while (sIn.available() > 0)
      {
         line = sIn.readLine();

         // if OMCLOSE was successful,then clear out display
         if (line.equals("OMCLOSEOK"))
         {
             firstUpdate = 0;
             xcount = 0;
             frame.clearTrendDisplay();
         }
         else if( line.startsWith("OMUPDATE") )
         {
             String value;
             String name;
             StringTokenizer st = new StringTokenizer(line, " ;=");

             // First - get rid of the OMUPDATE token
             name = st.nextToken();

             // Now - get each name-value pair that is on the OMUPDATE line
             while ( st.hasMoreTokens() )
             {
                name = st.nextToken();      // Should be NAME token
                value = st.nextToken();     // Should be VALUE token
                frame.newNVpair(name, value);
             }
             firstUpdate = 1;
         } // if OMUPDATE
      } // end of if available

      // Only start drawing when we get the first update!!
      if( firstUpdate > 0 )
      {
        frame.displayTrend();
        xcount++;;
      }

   } catch (IOException e) {}
}

/////////////////////////////////////////////////////////////////
// METHOD:   sSend
//
// FUNCTION:
//   -- send a message to the server offering OM data
//
/////////////////////////////////////////////////////////////////
public void sSend(String message)throws IOException
{
   if( message.charAt(message.length()-1) == '\n' )
     sOut.writeBytes(message);
   else sOut.writeBytes(message + "\n");
}
}   // END OF CLASS TCPClient
```

- Appendix Page 12 -

```
/*
// trendmlib.c
// The Foxboro Company Confidential
// Copyright (c) The Foxboro Company.  All Rights Reserved
// G. Couper 8/14/96
*/


/*
include "om_user.h"
Don't use this one.  It calls ipc.h which calls rpc/types.h
which declares bool_t in conflict with StubPreamble.h
*/

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

#include "psap.h"
#include "om_udef.h"
#include "om_udat.h"

#include <StubPreamble.h>   /* for conversion of java structs to C */
#include <javaString.h>     /* string manipulation */
#include "ThreadedTrendServer.h"  /* specially generated from trendmserver.java */

#include "om_ecode.h"

#define NO_IMPORT   FALSE
#define NO_SUSPEND  FALSE
#define SUSPEND     TRUE



long
ThreadedTrendServer_omlistcreate ( struct HThreadedTrendServer *this )
{
    HArrayOfByte*                tmp = unhand(this)->jin_om_desc;            /* find the java
buffer for the om_header */
    struct om_header_node*  in_om_desc = (struct om_header_node *) unhand(tmp)->body; /* find the
start of the om_header within that structure */

    HArrayOfByte*                tmp5 = unhand(this)->jin_net_adr_tbl;       /* find the java
buffer */
    struct net_adr* in_net_adr_tbl = (struct net_adr *) unhand(tmp5)->body; /* find the start of
the body within the buffer */

    HArrayOfByte*                tmp2 = unhand(this)->jin_var_list;          /* find the java
buffer */
    struct open_var*  in_var_list = (struct open_var *) unhand(tmp2)->body; /* find the start of
the body within the buffer */

printf(" c: omlistcreate - creating the list...\n");


    unhand(this)->jlistpoints = 0;                              /* record that we have no point in
the list */

    in_om_desc->task_status = OM_R_ACCESS;
    in_om_desc->net_adr_tbl_ptr = in_net_adr_tbl;
    in_om_desc->size_net_adr_tbl = 3;
    in_om_desc->open_list_ptr = in_var_list;
    in_om_desc->cur_size_open_list = 3;

}



long
ThreadedTrendServer_omlistadd ( struct HThreadedTrendServer *this, struct Hjava_lang_String
*Jname )
{
```

```
/*
This routine adds a new point name the the om list.
*/
    char Cname[100];

    HArrayOfByte*           tmp2 = unhand(this)->jin_var_list;  /* find the java buffer */
    struct open_var*  in_var_list = (struct open_var *) unhand(tmp2)->body ; /* find the start of
the body within the buffer */

    in_var_list += unhand(this)->jlistpoints ;   /* add it in the correct place on the list */


    javaString2CString( Jname, Cname, sizeof(Cname) );
    printf(" c: omlistadd - adding to list. Name is %s\n", Cname);

    strcpy(in_var_list->name,        Cname );
          in_var_list->var_desc = NOTIFY;
          in_var_list->delta    = 0.1;

    unhand(this)->jlistpoints++;   /* bump the count for next time */
}




long
ThreadedTrendServer_omlistopen ( struct HThreadedTrendServer *this )
{
    HArrayOfByte        *tmp = unhand(this)->jin_om_desc;                 /* find the java buffer for
the om_header */
    struct om_header_node  *in_om_desc = (struct om_header_node *) unhand(tmp)->body; /* find the
start of the om_header within that structure */

    HArrayOfByte*           tmp2 = unhand(this)->jin_var_list;      /* find the java buffer */
    struct open_var*  in_var_list = (struct open_var *) unhand(tmp2)->body ; /* find the start of
the body within the buffer */

    HArrayOfByte*           tmp5 = unhand(this)->jin_net_adr_tbl; /* find the java buffer */
    long*                   tmp6 = (void *) unhand(tmp5)->body;   /* find the start of the
body within the buffer */
    long*                   tmp7 = (void *) *tmp6;                /* find the start of the
body within the buffer */
    struct net_adr*  in_net_adr_tbl = (void *)  tmp6;             /* find the start of the
body within the buffer */

    HArrayOfByte*           tmp3 = unhand(this)->jin_data;                /* find the java buffer
*/
    struct value*     in_data_ptr = (struct value *) unhand(tmp3)->body ; /* find the start of
the body within the buffer */

    int rtn;
    int i;


    rtn = omopen(in_om_desc, (int *) &unhand(this)->jin_open_id);
    printf(" omopen returns = %x\n", rtn);



    /*
     * if (( in_data_ptr = (struct value *)v_varlist (8)) == NULL)  // this is the contruct that
works in the example
     * in_data_ptr = (struct value *) v_varlist(3);                 // this one does not work, but
should
     */

    if ( in_data_ptr == NULL)
    {
    printf(" Can't allocate space to receive updates.\n");
    (void) ThreadedTrendServer_omlistclose ( this );

    }
```

- Appendix Page 14 -

```
    sleep(11);

    printf("Open id = %d ... ", unhand(this)->jin_open_id );

    rtn = omread(unhand(this)->jin_open_id, 3, in_data_ptr);
    if (rtn != OM_SUCCESS)
    {
        printf("omread return = %d\n", rtn);
        omclose(unhand(this)->jin_open_id, in_om_desc, in_var_list, in_net_adr_tbl);
    }
    else
    {
        for (i = 0; i<3; i++)
            {
            printf("Variable [%d] = %f\n", i,in_data_ptr->uval.fpoint);
            in_data_ptr++;
            }
    }

}




long
ThreadedTrendServer_omlistgetupdate( struct HThreadedTrendServer *this,
                        HArrayOfByte* OutBuf,
                        long count )
{
    HArrayOfByte*           tmp2 = unhand(this)->jin_var_list;  /* find the java buffer */
    struct open_var*  in_var_list = (struct open_var *) unhand(tmp2)->body ; /* find the start of
the body within the buffer */

    HArrayOfByte*           tmp3 = unhand(this)->jin_data;               /* find the java buffer
*/
    struct value*    in_data_ptr = (struct value *) unhand(tmp3)->body ; /* find the start of
the body within the buffer */

    int   rtn;
    int   i;
    int   numvars;
    char  *data = unhand(OutBuf)->body;
    int   len     = obj_length(OutBuf);
    char  my_data[100];
    int   actual;
    pid_t pid;


    if (len < count)
    {
        actual = len;
    }
    else
    {
        actual = count;
    }

    /* request update data */
    pid = getpid();

    /* get ready for update data */
    numvars = 0;
    strcpy( data, "");


    rtn = dqchange(pid, NO_SUSPEND, (int *) &unhand(this)->jin_open_id, 3, in_data_ptr,
&numvars);

    if (numvars > 0)
    {
        strcpy( data, "OMUPDATE ");
        for (i = 0; i<=(numvars-1); i++)
            {
```

```
        sprintf(my_data, " %s = %f ;", in_var_list[in_data_ptr->index].name
,in_data_ptr->uval.fpoint);
        strcat( data, my_data );
        in_data_ptr++;
        }
    strcat( data, "\n" );
    printf(data);
    }

    return strlen(data);
}




long
ThreadedTrendServer_omlistclose ( struct HThreadedTrendServer *this )
{
    HArrayOfByte                    *tmp = unhand(this)->jin_om_desc;  /* find the java buffer for
the om_header */
    struct om_header_node *in_om_desc = (struct om_header_node *) unhand(tmp)->body; /* find the
start of the om_header within that structure */

    HArrayOfByte*           tmp2 = unhand(this)->jin_var_list;         /* find the java buffer */
    struct open_var*  in_var_list = (struct open_var *) unhand(tmp2)->body ; /* find the start of
the body within the buffer */

    HArrayOfByte*                   tmp5 = unhand(this)->jin_net_adr_tbl; /* find the java buffer */
    struct net_adr*  in_net_adr_tbl = (struct net_adr *) unhand(tmp5)->body;              /*
find the start of the body within the buffer */

    int rtn;

    printf(" c: Closing list... ");
    rtn = omclose(unhand(this)->jin_open_id, in_om_desc, in_var_list, in_net_adr_tbl);
    printf(" Return was %d\n", rtn);
    return ( (long)rtn );
}
```

```java
//
// trendmserver.java
// The Foxboro Company Confidential
// Copyright (c) The Foxboro Company.  All Rights Reserved
// G. Couper, B. Canna  8/14/96
//
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;
import java.applet.*;

public class trendmserver extends Applet
{
    public static void main(String arg[] ) throws IOException
    {
        ServerSocket s =(ServerSocket) null;
        Socket        sock;
        int           i = 0;

        System.out.println("\ntrendmserver: Waiting for a socket...");
        try
        {
            s = new ServerSocket( 4322 , 60 );  // port/socket#, seconds before timeout

            while( true )  // this server's work is never done.
            {
                System.out.println("trendmserver: Waiting for a client...");
                sock = s.accept();  //accecpt a connection

                System.out.println("trendmserver: Connection accepted. Spawning new thread [" + i +
"]." );

                new ThreadedTrendServer( sock, i ).start();

                i++;
            }//end of while to get next client

        }//end of new socket try
        catch(IOException e)
        {
            System.out.println("trendsmserver: Quitting because of error = " + e);
        }//end of new socket catch
    }//end of main routine
}//end of class trendmserver
```